COMP 251: Data Structures and Algorithms Winter 2006

Lecture on Recurrences with Full History

Lecturer/TA: Ethan Kim *Scribe:*

1 Overview

In previous lectures, we stated the Master Theorem without proof. In this lecture, we discuss a relaxed version of the Master Theorem, namely Divide and Conquer Relations, with a full proof. Furthermore, we discuss a technique to solve recurrence relations with full history, which can be used to analyze the expected (average) running time of quick sort.

2 Divide and Conquer Relations

In a divide–and–conquer algorithm, the problem is divided into smaller subproblems, each of which is solved recursively, and then the solution to the subproblems are combined to solve the original problem.

Suppose our algorithm divides the original problem into a subproblems, each of size $1/b$ of the original problem. And suppose further that the algorithm combining the subproblems runs in time cn^k. (Note: a, b, c, k are constants.) Then, the running time of the algorithm is recursively defined as:

$$T(n) = aT(n/b) + cn^k \tag{1}$$

Then, we have the following theorem:

Theorem 1. *The solution of the recurrence relation $T(n) = aT(n/b) + cn^k$, where a and b are integer constants, $a \geq 1, b \geq 2$ and c and k are positive constants, is*

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } a > b^k \\ O(n^k \log n) & \text{if } a = b^k \\ O(n^k) & \text{if } a < b^k \end{cases}$$

Proof. Assume, for simplicity, that $n = b^m$, so that n/b is always an integer. We first try to expand the recurrence a few times:

$$T(n) = a(aT(n/b^2) + c(n/b)^k) + cn^k = a(a(aT(n/b^3) + c(n/b^2)^k) + c(n/b)^k) + cn^k$$

If we expand all the way to $n/b^m = 1$, we get:

$$T(n) = a(a(\cdots T(n/b^m) + c(n/b^{m-1})^k) + \cdots) + cn^k$$

Let's assume that $T(1) = c$. Then,

$$T(n) = ca^m + ca^{m-1}b^k + ca^{m-2}b^{2k} + \cdots + cb^{mk}$$

which implies that

$$T(n) = c\sum_{i=0}^{m} a^{m-i}b^{ik} = ca^m \sum_{i=0}^{m}(\frac{b^k}{a})^i$$

This is a geometric series. We have the following three cases:

Case 1: $a > b^k$
In this case, the factor of the geometric series is less than one, and thus the series converges to a constant even if m goes to infinity. Therefore, $T(n) = a^m$. Since we assumed $m = \log_b n$, we have that $T(n) = a^{\log_b n} = n^{\log_b a}$.

Case 2: $a = b^k$
In this case, the factor of the geometric series is 1, and thus $T(n) = O(a^m m)$. Since $a = b^k$, $k = \log_b a$. As before, $a^m = n^{\log_b a} = n^k$, and $m = \log_b n$. Thus, we have $T(n) = O(n^k \log n)$.

Case 3: $a < b^k$
In this case, the factor of the geometric series is greater than 1. Denote b^k/a by F. Using the standard expression for summing a geometric series:

$$T(n) = ca^m \frac{F^{m+1} - 1}{F - 1} = O(a^m F^m) = O((b^k)^m) = O((b^m)^k) = O(n^k)$$

$\square$

Question: What's the difference between this theorem and the Master Theorem? (In Master Theorem, there were *gaps* between two cases where one cannot use the master theorem.)

3 Recurrence Relations with Full History

A **full-history recurrence relation** is one that depends on all the previous values of the function, not just few of them. For instance, the recurrence relation from *merge-sort* is:

$$T(n) = 2T(n/2) + O(n)$$

Here, $T(n)$ only depends on $T(n/2)$. Let's look at another example :

Example 1

$$T(n) = c + \sum_{i=1}^{n-1} T(i) \tag{2}$$

where c is a constant, and $T(1)$ is given. In this example, $T(n)$ depends on all the previous values $T(i)$, $i = 1 \ldots n - 1$, at each level of recursion. Now, let's try to solve this recurrence using the **shifting and canceling technique**. The key idea is to convert the full-history recurrence formation into a recurrence that depends on only a few previous values. Rewriting the recurrence (2) for $n+1$:

$$T(n+1) = c + \sum_{i=1}^{n} T(i) \tag{3}$$

If we subtract (2) from (3), we get $T(n+1) - T(n) = T(n)$. So, $T(n+1) = 2T(n)$, implying that $T(n+1) = T(1)2^n$. It holds for the base case $T(1)$, and by induction, the formula $T(n+1) = T(1)2^n$ holds!!

...Really? For example, set $T(1) = 1$ and $c = 5$, $T(2) = 5 + 1 = 6 \neq 2T(1)$. The fault is at picking the incorrect base case. The proof does not work for $T(2)$, since $T(1)$ is not necessarily canceled by c. To solve correctly, note that $T(2) = T(1) + c$(by definition), and the proof is correct for all values $n \geq 2$. Thus, $T(n+1) = (T(1) + c)2^{n-1}$.

Let's look at a little more complicated example:

Example 2

$$T(n) = n - 1 + \frac{2}{n} \sum_{i=1}^{n-1} T(i) \text{ (for } n \geq 2). \qquad T(1) = 0. \tag{4}$$

As before, we use the shifting and canceling terms technique. Rewriting the recurrence for $n+1$:

$$T(n+1) = (n+1) - 1 + \frac{2}{n+1} \sum_{i=1}^{n} T(i) \qquad (n \geq 2). \tag{5}$$

To remove the fractions in front of the series, multiply both sides of (4) by n, and both sides of (5) by $n+1$:

$$nT(n) = n(n-1) + 2\sum_{i=1}^{n-1} T(i) \qquad (n \geq 2). \tag{6}$$

$$(n+1)T(n+1) = (n+1)n + 2\sum_{i=1}^{n} T(i) \qquad (n \geq 2). \tag{7}$$

Now, subtract (6) from (7) to obtain

$$(n+1)T(n+1) - nT(n) = (n+1)n - n(n-1) + 2T(n) = 2n + 2T(n) \ (n \geq 2),$$

which implies that

$$T(n+1) = \frac{n+2}{n+1} T(n) + \frac{2n}{n+1} \qquad (n \geq 2).$$

This recurrence is easier to solve. To make things even simpler, substitute 2 for $\frac{2n}{n+1}$, and get a close approximation:

$$T(n+1) \le \frac{n+2}{n+1}T(n) + 2 \qquad (n \ge 2). \tag{8}$$

Now, expand (8) to get

$$\begin{aligned}
T(n) &\le 2 + \frac{n+1}{n}\left(2 + \frac{n}{n-1}\left(2 + \frac{n-1}{n-2}\left(\cdots\frac{4}{3}\right)\right)\right) \\
&= 2\left(1 + \frac{n+1}{n} + \frac{n+1}{n}\frac{n}{n-1} + \frac{n+1}{n}\frac{n}{n-1}\frac{n-1}{n-2} + \cdots + \frac{n+1}{n}\frac{n}{n-1}\frac{n-1}{n-2}\cdots\frac{4}{3}\right) \\
&= 2\left(1 + \frac{n+1}{n} + \frac{n+1}{n-1} + \frac{n+1}{n-2} + \cdots + \frac{n+1}{3}\right) \\
&= 2(n+1)\left(\frac{1}{n+1} + \frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{3}\right) \\
&= 2(n+1)(H(n+1) - 1.5),
\end{aligned}$$

where $H(n) = 1 + 1/2 + 1/3 + \cdots + 1/n$ is the Harmonic series. The Harmonic series has an approximation, $H(n) = \ln n + \gamma + O(1/n)$, where $\gamma = 0.577..$ is **Euler's constant**. Hence the solution for $T(n)$ is

$$T(n) \le 2(n+1)(\ln n + \gamma - 1.5) + O(1) = O(n \log n).$$

3.1 Application to Quick Sort

And, this recurrence turns out to be useful for the analysis of expected running time of *quick sort*. Suppose we are given an array of n elements, and the ith smallest element is the pivot. Then, the running time of quick sort is given by the recurrence:

$$T(n) = n - 1 + T(i-1) + T(n-i)$$

$n-1$ comparisons are needed to partition, and the two subproblems of sizes $i-1$ and $n-i$ are solved recursively. Now, if we assume that each element has the same probability of being selected as the pivot, then the average running time is:

$$\begin{aligned}
T(n) &= n - 1 + \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i)) \\
&= n - 1 + \frac{1}{n}\sum_{i=1}^{n}T(i-1) + \frac{1}{n}\sum_{i=1}^{n}T(n-i) \\
&= n - 1 + \frac{2}{n}\sum_{i=0}^{n-1}T(i).
\end{aligned}$$

This is the recurrence (4) we studied. Hence, the expected (average) running time of quick sort is indeed $O(n \log n)$.